A Topology Reconfiguration Mechanism for Distributed Computer Systems

M. Bozyigit and Y. Paker
The Polytechnic of Central London, 115 New Cavendish Street, London W1M 8JS, UK

A topology reconfiguration algorithm for a densely distributed computer structure composed of computers of low cost/performance ratio (e.g. microcomputers) is given. A distributed routing mechanism based on Baran's 'hot potato' heuristic routing algorithm is discussed. This mechanism is applied to a multicomputer system (VTM—Variable Topology Multicomputer system) proposed to be reconfigurable according to the application requirements. An analysis of the algorithm regarding the initialization of a distributed computer system is given and the preliminary system performance results in applying this algorithm are shown. The results of a run time implementation are also compared with those of a traffic-load balancing algorithm which is basically a static fixed routing algorithm, suggested by the authors for densely and uniformly distributed computer systems.

INTRODUCTION

Densely distributed computer systems constructed of limited performance and low cost computers need non-elaborate, easily implementable routing mechanisms, whereby the information flows with minimum control overhead. Fixed routing mechanisms which can utilize optimum or nearly optimum paths between the nodes of a distributed system require little overhead. Once a path is assigned it remains fixed during the running period. The authors have shown that for homogeneous, dense and initially uniform computer networks the fixed paths can be assigned such that the uniformity in the internal run time traffic can be achieved.[1] On the other hand, the fixed routing is rigid and does not allow dynamic reconfigurability of the system involved, which becomes significant in a distributed processing environment.

An important requirement in distributed computer systems (DCS) is the survivability (allied to reliability) in case of link, or node, or link and node failures. It is normally possible, assuming single fault case, to determine a new path where the old path has permanently failed, provided that there is at least one possible new path. The heuristic shortest path algorithms, well-known for their versatility, need complete topology information to be dynamically available for this purpose. This method becomes computationally time-consuming and requires high storage capacity at each node. A reliable system needs an effective automatic mechanism to recover from a failure. The system must be able to adapt itself to the possible configurational changes in a finite time. The routing mechanisms such as (1) random, (2) flooding, (3) ideal observer, and (4) adaptive, can respond dynamically to the changes. For the first two, a node is not necessarily aware of the full system configuration. All available nodes and links are used, to get to the destination. The third method needs a system control centre which is ideally assumed to watch the entire system and be responsible for incorporating configurational changes as well as the traffic flow conditions. The last technique is basically a flow control mechanism which can also respond to the topology changes in terms of some delay function.

In this paper a reconfiguration and routing mechanism is proposed which is implemented on a proposed multicomputer system to achieve a degree of reliability and maintain uniformity in initial traffic load distribution. The technique employs different routing criteria for the data and the configuration control messages, similar to ARPANET's New Routing Algorithm.[2,3] The method is deterministic and based on Baran's 'hot-potato' algorithm.[4,5] The performance thus experienced is compared with another deterministic traffic-load balancing routing mechanism based on a shortest path finding algorithm, under similar application requirements.[6,7]

IMPLEMENTATION ASPECTS

Given a network of bi-directional communication links it first undergoes an initialization phase whereby the nodes exchange the topology information until all the nodes are aware of the system configuration. A change in topology is indicated by an information unit which is called topology message (TM).

Each node maintains an $n \times m$ distance table (D) where n is the size of the DCS involved and m is the connectivity. An entry $d(i, j)$ at a node k indicates the distance from node k (host for D) to node i via the jth neighbour. The distance table at a node k is associated with a routing table (R). An entry $r(i, j)$ at a node k is the identification of the jth node next (adjacent) to k on a shortest path from k to i.

A node is assumed failed or inactive if it is not accessible from any other node in the system. This manifests itself as the failure of all the connections with the neighbouring nodes. The functions that a node is to carry out for this algorithm are to: (1) detect existing connections by sensing successful completion of line handshake procedures; (2) update the distance table to reflect the current configuration; (3) prepare and send the topology messages to indicate a change in configuration; (4) receive topology messages and interpret the topology changes; (5) record the shortest paths in the routing table.

Initially a node detecting a new adjacent node confirms

CCC-0010-4620/82/0025-0087 $03.00

the changes by updating the proper entry of the distance table. For each new neighbour a topology message is formed and broadcast to other neighbours. Meanwhile the new neighbour receives the full account of the connection information available at the detecting node. A topology message (TM) traversing through the system will come to a stop at a node if it does not change the shortest distance to the end-node to which the change is directly related. Otherwise, it causes further TMs one for each neighbouring node.

The reconfiguration algorithm consists of three sub-algorithms; (1) new link detection and handling, (2) failed link handling, (3) processing incoming topology messages.

The three algorithms used are given in the Appendix.

ROUTING MECHANISM

The routing information is compiled from the distance tables depending on the routing mechanism to be employed.

The mechanism adapted here utilizes a round-robin routing technique whereby the shortest paths are allocated to the incoming traffic in turn. The row i of routing table at a node k records the equal distance shortest paths between the nodes k and i. Each path is identified by the node on the other end of the line. The number of equal distance paths cannot be greater than the number of neighbour nodes.

Assume that i is the ultimate destination for an information message arriving at a node k, $c(i)$ is the number of shortest paths from k to i, and j is the path selection index initially set to zero at the system start. Then the routing procedure for path selection is given as follows:

(1) fetch the identification of destination node $i :=$ destination field of the TM
(2) path selection control
if $j = 0$ then $j := c(i)$;
(3) determine the next node adjacent to k on the selected path
$l := r(i, j)$;
update path selection counter
$j := j - 1$;
(4) send the message for l;
(5) exit.

Dynamic reconfigurability achieves high overall system reliability. During the system running each topological change detected is soon incorporated in the system by means of exchange of relevant topology messages. The counter-effect of high reliability is the overhead involved as compared to a fixed non-reconfigurable case where no change can take place. In fact, intensive reconfiguration requests can degrade the system extensively, since the processing power is then shared between the information and topology messages.

SIMULATION RESULTS

The reconfiguration algorithm is simulated on a VTM (Variable-Topology-Multicomputer) system which is proposed to be reconfigured to meet the requirements of an application.[8] The system is basically a dense, uniform, and low cost computer communication network. A node is composed of a local computer (LC), a communication computer (CC), an input port (IP), and an output port (OP) as shown in Fig. 1.

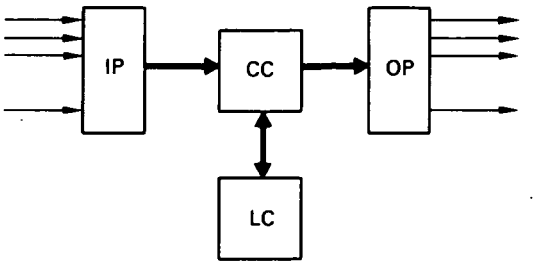

Figure 1. Node computer.

The message transmission requests are externally generated at LCs and put on the communication subnetwork. The application is simulated such that interarrival times of the communication requests between the nodes follow exponential distribution and transmission handling time on a link is constant throughout the system.

The system undergoes an initialization phase before the actual application run takes place. Assuming that the DCS consists of n nodes forming a cross-connected closed topology we can derive an analytical approximation for the total number of messages to be exchanged during the initialization which is an indication of overhead involved against the benefits of the reconfigurability achieved.

The node detecting a new live link sends the full topology information available to the node on the other end of that link, in the form of $n - 1$ messages which are referred to as topology messages (TM). Each message, at this stage, carries the shortest distance information to a particular node from the detecting node. At an m link node, for each new link a message is broadcast to the remaining $m - 1$ links. Thus, each link during the first stage of initialization would have $(m - 1) + (n - 1)$ TMs waiting on it for transmission. For m links $m(n + m - 2)$ TMs will be generated at each node.

A TM regarding a particular node is terminated if the existing topology information on that node has not been affected. Otherwise a new TM is formed to broadcast the change. The total number of TMs depends on how soon the system becomes aware of the full topology. In the case of detection of a new node it involves the accessibility of this node from all other nodes in the system. There are $n - (m + 1)$ such nodes per new node. Thus total number

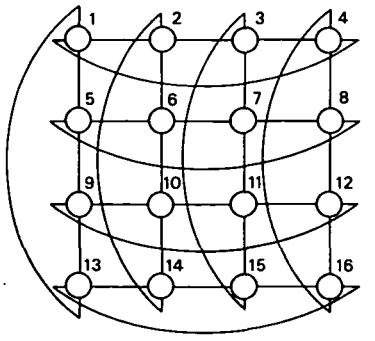

Figure 2. A 16-node network.

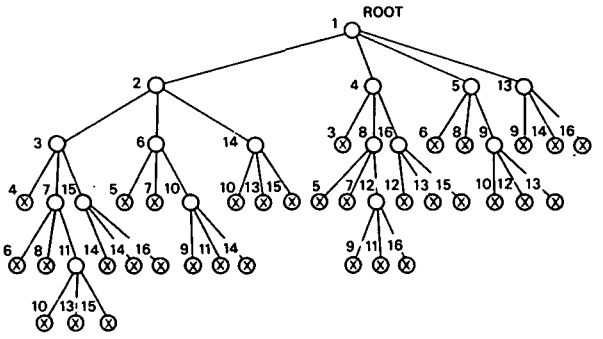

Figure 3. Topology message exchange tree.

of TMs is equivalent to the number of TMs of paths of length 2 or more which, in terms of number of links, can be given as $\sum_{i=2}^{h} P_i$ where P_i is the number of paths of length i to/from the new node, and h is the length of the longest path.

On a path at an intermediate node $m-1$ new TMs are formed and routed another step forward to indicate the changes. This process is repeated until the terminal node on the path is reached, where the configuration information is no longer affected. To illustrate, the transmission takes place layer after layer as shown in Fig. 3 related to the topology shown in Fig. 2.

The number of TMs required to establish the shortest paths to the new node becomes

$$(m-1)\sum_{i=2}^{h} P_i \qquad (1)$$

per node provided that the first TM arriving at a node carries the shortest path information.

The total number of TMs created by the incorporation of the new node is then

$$\text{TMnode} = m(m+n-2) + (m-1)\sum_{i=2}^{h} P_i \qquad (2)$$

This equation represents a lower limit for the total number of TMs.

Often the first TM arriving at a node does not carry the shortest path information, either because of the queueing effect or the TM handling sequence employed. For example, if the handling procedure was such that the TMs related to a live link are handled and routed before attempting the next link waiting for detection then Eqn (1) becomes

$$(m-1)\sum_{i=1}^{m}\sum_{j=2}^{h} P_{ij} \qquad (3)$$

where P_{ij} indicates the number of paths of length j needed for a node to be aware of a change, h indicates the length of the longest shortest path travelled; provided that $i-1$ links have previously been detected and handled.

Expression (3) cannot take smaller values than Expression (1) since it can be opened up as

$$(m-1)\sum_{j=2}^{h} P_{ij} + (m-1)\sum_{i=2}^{m}\sum_{j=2}^{h} P_{ij} \qquad (4)$$

where the first term is equal to Expression (1), providing the proof that Expression (3) is greater than that of (1).

Assuming that the entire system comes up simultaneously and the links are handled accordingly then Expression (2) becomes

$$\text{TMsystem} = n\left[m(m+n-2) + (m-1)\sum_{i=2}^{h} P_i\right] \qquad (5)$$

After replacing the number of shortest paths by $(n-m-1)$ we have

$$\text{TMsystem} = n(n-1)(2m-1) \qquad (6)$$

Equation (6), when applied to the network in Fig. 2 gives 1680 for TMs. This is a lower limit to the total number of TMs that can be generated to complete the initialization.

The simulation model, on the other hand, which reflects a real situation, gave a total of 1776 TMs. The result is as expected. The simulation model is constructed under the same assumptions used for theoretical derivations.

The total number of TMs becomes important as a performance factor, especially because of system settlement time and the buffer allocation requirements which are, similarly, functions of system size, connectivity, rules for handling configurational changes, and the communication protocols.

The reconfiguration rules proposed in this work suggest that all the TMs related to the changes are generated first, followed by transmission in a strict order whereby the receiving nodes become aware of the changes in the same order. This technique appears to provide fewer TMs, shorter settlement time (ST) but longer output buffer requirements.[1] The 16-node cross-connected topology system needed as high as 19 TMs to be stored per detected live link before the transmission is attempted. The length of the ST is tied to the processing speed of the processors involved as well as other factors such as buffer lengths and communication protocols. Since any specific timing figures require the selection of a particular processor such quantifications are deliberately avoided. For comparative analysis see Ref. 1.

The reconfiguration algorithm has provisions for link/node failures/coming-ups which can be incorporated during the course of normal operation. The settlement time incurred for such changes is closely connected to the level of the existing traffic. High traffic levels dictate longer delays for TMs to flow through due to the queueing effect unless such messages are given higher priority.

PERFORMANCE OF THE ALGORITHM

The second phase, after initialization, is the actual network operation when subject to a given traffic flow. The routing mechanism is based on the utilization of equal length shortest paths assigned during the initialization, in a round-robin fashion.

The cumulative distribution of the message response time is as shown in Fig. 4 where the vertical axis represents the cumulative relative frequency. This result is very near to that of the traffic-load balancing fixed routing algorithm discussed in Ref. 1. That algorithm, however, does not accommodate any configurational changes. The approach taken is based on the preference of the least distance path while establishing a balance in

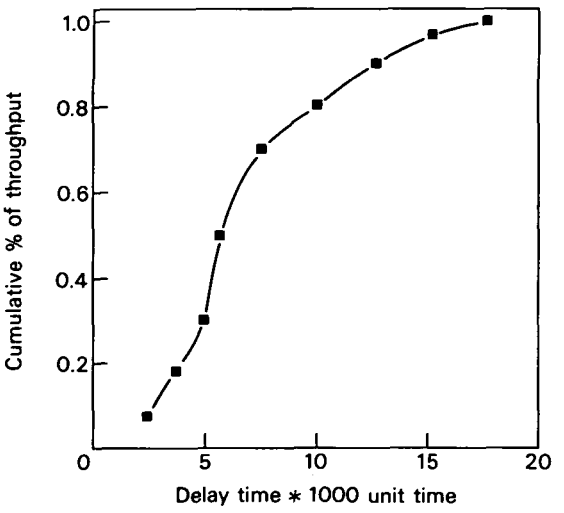

Figure 4. Distribution of message delay.

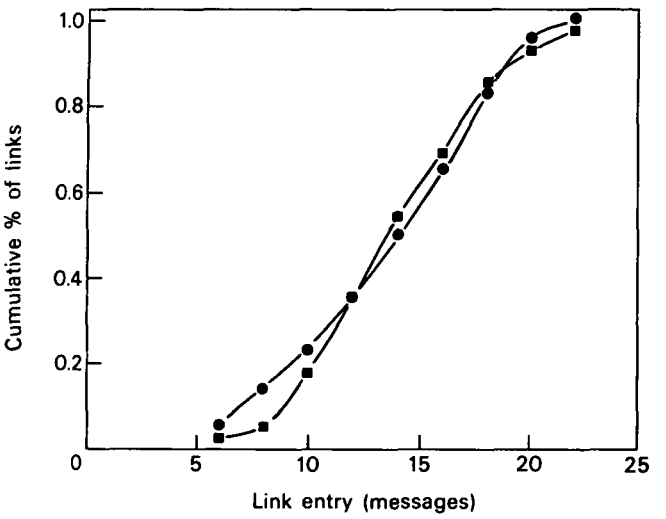

Figure 5. Cumulative distribution of link entries: ■ round-robin (reconfigurability); ● fixed (balanced).

Table 1. Performance data of reconfiguration

	Non-failure	A link failure	A node failure	A node comes up
External input in number of messages	477	477	465	442
Throughput in number of messages	469	461	437	396
Delay in unit time	4783	6319	6613	7146
Buffer length in number of messages	1.28	1.60	1.70	3.50

traffic flow provided that there are at least one or more shortest paths between node pairs.

The distribution of link entries shows that the round-robin routing is slightly better in terms of uniformity in link utilization with a standard deviation from the mean of 3.6 against 5.6, at an average link load of 15 messages (see Fig. 5). In addition it provides alternative paths and system reconfigurability dynamically.

The disadvantage of such a reconfigurable algorithm is the overhead incurred during the actual system operation, because of reconfigurations. The performance data for three separate operations is tabulated in Table 1. The first column is non-failure case. The second column is link failure and the third is the node failure with all its link to the external world down. The major configurational changes such as node failure or node return are the source of longer response time, lower throughput, and longer buffer space requirements. The last column shows the case where an inactive node comes up after a certain time during the system operation. As expected, heavier traffic load (application + reconfiguration) can cause system degradation. In the case of a node coming up more TMs need to be exchanged since the new node also needs the reconfiguration tables to be built up.

CONCLUSION

The reconfiguration algorithm discussed in this paper achieves a degree of uniformity regarding the distribution of the internal traffic flow in homogeneous and dense network type DCS. The simulation model showed that this mechanism is successfully applicable to DCS. In fact, it is inherently a distributed algorithm.

The alternatives to this algorithm vary from fully broadcast to fully dynamic adaptive techniques. The former needs to duplicate all the messages and the latter needs a routing function to be computed for each message arrived. Therefore the former can require unnecessarily high buffer space utilization and the latter, on the other hand, can demand high processing power to control the flow with all the tables it needs to maintain. The reconfiguration algorithm which is adaptive in nature but designed for configurational changes only, is a deterministic algorithm.

The ARPANET's New Routing Algorithm[2,3] has a significant resemblance to the reconfiguration algorithm discussed in this paper. The routing technique employed for the exchange of update information, in both cases, is basically a 'flooding' mechanism. In ARPANET's algorithm the update information based on the link delays is periodically (or non-periodically for up/down states) exchanged, thus affecting the routing decision taken at individual nodes. This requires, at each node, the maintenance of an up-to-date data base which contains delay information over each link in the network.

The reconfiguration algorithm discussed in this paper employs routing tables updated by the information received from the immediate neighbours only. A topological change related to the state of a link is propagated until it has no effect on the routing decision-taking process. Summarizing the above, this technique has the following limitations: (1) the system adapts to the topological changes but not to the congestion of any sort; (2) data routing mechanism is based on the shortest path information made available by the exchange of the

reconfiguration information; (3) it is assumed that a topological change regarding a link in the network is reflected at a node only after the previous change related to the same link has been handled. The frequent transitory changes may cause inconsistent update of routing tables, especially if they are not handled in the order they are generated. Such changes can also be a source of unnecessary activities, if they are not accounted for.[3]

An implementation of this algorithm on an Intel 8080 microcomputer covering downs (failures) and comings-ups (repairs), required under 300 instructions only. In a microprocessor based distributed computer environment this algorithm could provide effective implementation of transaction exchange, meanwhile establishing reliability and reconfigurability.

Acknowledgements

We express our thanks to the referee for his helpful comments on the relevance of this algorithm including its limitations, to the ARPANET's New Routing Algorithm.

REFERENCES

1. M. Bozyigit, A Dense Variable Topology Multicomputer System: Specifications and performance. PhD Thesis, Polytechnic of Central London (1979).
2. J. M. McQuillan, I. Richer and E. C. Rosen, An overview of the new routing algorithm for the ARPANET, *Proceedings of the Sixth Data Communication Symposium*, Pacific Grove, California (November 1979).
3. E. C. Rosen, The updating protocol of ARPANET's New Routing Algorithm. *Computer Networks* **4** (No. 1), 11–19 (February 1980).
4. P. Baran, On distributed communication networks. *Institute of Electrical and Electronics Engineers, Transactions, Communications Systems* **CS–12**, 1–9 (March 1964).
5. W. D. Tajibnapis, A correctness proof of topology information maintenance protocol for a distributed computer network. *Communications of the ACM* **20**, 477–485 (1977).
6. L. E. Hitchner, A comparative investigation of the computational efficiency of shortest path algorithms. *Oper. Res. Ctr. Rep.*, 66–75 (November 1968).
7. M. Bozyigit and Y. Paker, A fixed routing problem in large and high connectivity networks. *The Computer Journal* **22**, 246–250 (1979).
8. Y. Paker and M. Bozyigit, A variable topology multicomputer. *Euromicro 76*, 141–151 (1976).
9. A. T. Berztiss, *Data Structures: Theory and Practice*, Academic Press, London (1971).

Received December 1980

APPENDIX

Nomenclature for the reconfiguration algorithms

a = source node
b = destination node
c = node adjacent (neighbour) to node a
$d^a_{b_j}$ = distance between a and b via the jth neighbour of a (the superscript a is omitted wherever it is obvious)
r^a_b = the first neighbour on shortest path between a and b indicated by a neighbour of a on that path
TM^a_b = topology message compiled at a to be sent to b
$c \leftarrow TM^a_b$ = send TM^a_b to c
V^a_b = shortest distance between a and b
W^a_c = weight of link between a and c
X^a_i = ith neighbour of a
t = a temporary variable

Algorithm-1: Handles a new link coming up at node-a

1. (a new link detected at the local node a)
   link (a, c) becomes live;
2. (update distance and routing tables)
   $d^a_{c,j} := 1$ where jth neighbour = c,
   $V^a_c := 1$,
   $r^a_c := c$;
3. (prepare topology message)
   (set distance field of topology message)
   dist $(TM^a_c) := 1$,
   (set destination field of topology message)
   dest $(TM^a_c) :=$ c;
4. (send topology message to neighbouring nodes)
   $X^a_j \leftarrow TM^a_c$ for $j = 1,2,\ldots,m$ and $X^a_j \neq c$;
5. (send available topology information to new neighbour)
   (form topology message)
   dist $(TM^a_i) := V^a_i$,
   dest $(TM^a_i) := i$ for $i = 1,2,\ldots,n$.
   (send topology change messages to new neighbour)
   $TM^c_i := TM^a_i$;
6. Exit.

Algorithm 2: Handles link failures at node a

1. (a link failure detected at node a)
   link (a, c) failed;
2. (update the distance table)
   (save distance vector corresponding to failed link)
   $t_i := d^a_{i,j}$,
   (set vector $d^a_{i,j}$ to infinity)
   $d^a_{i,j} := \infty$ where $X^a_j = c$, and $i = 1,2,3,\ldots,n$;
   j = index of the failed link
3. (update routing tables and prepare topology messages)
   for $i :=$ from 1 to n with step 1 do
   if $V^a_i = t_i$ then
   $V^a_i := \min [d^a_{i,j}], j \in m$
   $r^a_i := X^a_j$ where $V^a_i = d^a_{i,j}$,
   $TM^a_i := V^a_i$,
   (send topology message).
   $TM^{x^a_j}_i := TM^a_i$ for $j = 1,2,\ldots,m$,
   except for $X^a_j = c$,
   fi,
   od;
4. Exit.

Algorithm 3: Handles arrived topology message at node a

1. (detect topology message at local node a)
   TM_b^c arrives at a;
2. (update distance and routing tables)
   (assign new distance for (a, b) path)
   $d_{b,j}^a := TM_b^c + W_c^a$,
   (find new min distance (a, b) path)
   $t_b := \min [d_{b,j}^a], j \in m$
   (assign the shortest path)
   if t_b not $= V_b^a$ then
   $r_b^a := X_j^a$ where $t_b = d_{b,j}^a$
   $V_b^a := t_b$,
   (prepare topology messages)
   dist $(TM_b^a) := V_b^a$,
   dest $(TM_b^a) := b^a$,
   (send topology message to neighbours)
   $X_j^a \leftarrow TM_b^a$ for all $j \in m$.
   fi;
3. Exit.

Algorithm 4: Round-robin routing algorithm

1. (input a message at a) TM_b^c arrives
2. (find the shortest path (c, b))
   if $c = (m - 1)$ then
   $c := 0$
   fi,
   (next node)
   $c := r_b^a$;
3. (message joins output queue)
   $OQ_c := TM_b^a$;
4. Exit.